Scheduling of Real-Time Systems

Contact
m.nasri@tudelft.nl
(appointment by email)
Online scheduling of periodic tasks

Buttazzo’s book, chapter 4


Disclaimer: A few slides have been taken from Giorgio Buttazzo’s website: 
http://retis.sssup.it/~giorgio/rts-MECS.html
Agenda

• Secrets behind L&L and hyperbolic bound tests

• RM schedulability **analysis**
  • Response-time analysis
  • Park test
  • A test for Harmonic task sets

• Lab 2 assignment description

• EDF schedulability **analysis**
  • Optimality of EDF for periodic tasks
  • EDF response-time analysis

If you want to pass the course, learn this lecture **very well**
Assumptions

• For this lecture, we assume
  
  **A1.** $C_i$ and $T_i$ are constant for every job of $\tau_i$
  
  **A2.** Tasks are fully preemptive
  
  **A3.** Context switch, preemption, and scheduling overheads are zero
  
  **A4.** Tasks are independent:
  
  • no precedence relations
  
  • no resource constraints
  
  • no blocking on I/O operations
  
  • no self suspension

Assume that tasks are indexed according to their priority ordering, namely,

$P_1 < P_2 < P_3 < \ldots < P_n$
Example

Is the task set feasible?
Yes $U \leq 1$

Does L&L test accept this task set?
No because $U > 2(2^{1/2} - 1) \sim 0.83$

Does hyperbolic bound test accept this task set?
No because $(0.4 + 1) \times (0.6 + 1) = 2.24 > 2$

Is the task set schedulable by RM? (think)

\[
\sum_{i=1}^{n} U_i \leq 1 \quad \text{necessary}
\]
\[
\sum_{i=1}^{n} U_i \leq n \left(2^{1/n} - 1\right) \quad \text{Liu and Layland test}
\]
\[
\prod_{i=1}^{n} (U_i + 1) \leq 2 \quad \text{Hyperbolic bound}
\]
Example

Is the task set schedulable by RM? (think)

Yes! Here is the schedule:

<table>
<thead>
<tr>
<th>$\tau_i$</th>
<th>$C_i$</th>
<th>$T_i$</th>
<th>$D_i$</th>
<th>$U_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>0.6</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>4</td>
<td>10</td>
<td>10</td>
<td>0.4</td>
</tr>
</tbody>
</table>

$U = 1$
Utilization-based tests and RM scheduling

\[
\sum_{i=1}^{n} U_i \leq 1
\]

EDF bound (necessary test for RM)

\[
\sum_{i=1}^{n} U_i \leq n\left(2^{1/n} - 1\right)
\]

L&L bound (sufficient test for RM)

\[
\prod_{i=1}^{n} (U_i + 1) \leq 2
\]

Hyperbolic bound (sufficient test for RM)

Bad news: We cannot have a better utilization-based test than the hyperbolic bound!

Under what circumstances the hyperbolic bound is better than the L&L test?

When the utilization values are far apart! E.g., \{0.9, 0.01\}

 TU Delft

Embedded and Networked Systems
Hyperbolic bound is tight

It is impossible to invent a new utilization-based test $A$ such that $A$ accepts a task set that the hyperbolic bound rejects!

In other words, as long as you only consider task utilizations into account, it is impossible to invent a test that is better than the hyperbolic bound!
Embedded and Networked Systems

Secrets behind
the Liu and Layland and hyperbolic bound tests

They found the
hardest-to-schedule task set with the minimal utilization:

- It is schedulable by RM
- It fully utilizes the processor if the execution time of any of the tasks is increased by $\epsilon$, then the task set is not schedulable anymore
- It has the minimum utilization among all task sets that fully utilize the processor

These task sets are on the boundary: by increasing any execution time, they will become unschedulable!

The smallest utilization that a barely schedulable task set can have, is our utilization threshold!

Exam hint: Pay attention to the definition of the hardest-to-schedule task set.

$$\sum_{i=1}^{n} U_i \leq n(2^{1/n} - 1)$$
$$\prod_{i=1}^{n} (U_i + 1) \leq 2$$
The hardest-to-schedule task set with the minimal utilization

- All tasks are released at the same time (no release offset)
- **Condition 1**: Periods follow $T_1 < T_2 < \ldots < T_n < 2T_1$
- **Condition 2**: Execution times follow:
  
  $C_1 = T_2 - T_1$
  $C_2 = T_3 - T_2$
  $C_i = T_{i+1} - T_i$
  $C_n = 2T_1 - T_n$

Why this generates a fully-utilized task set?

Exam hint: Remember these conditions for the exam.
Proof highlights

Goal:
Prove that the hardest-to-schedule task set (defined in the previous slide) has the minimum utilization among all other task sets that fully utilize the processor.

Step 1: prove the condition on periods

Claim: In a fully-utilized task set with the minimum utilization, periods must follow
\[ T_1 < T_2 < \cdots < T_n < 2T_1 \]

Step 2: prove the condition on execution times
(assuming that the condition on periods hold)

Assumption: given a set of periods \( T_1 < T_2 < \cdots < T_n < 2T_1 \)
Claim: the utilization of a task set that fully utilizes the processor will be minimum only if \( C_i = T_{i+1} - T_i, \ C_n = 2T_1 - T_n \)

This slide is just for your information (not in the exam). The interested students can follow the whole proof in the extra slides.
Deriving the hyperbolic bound

The genius trick:

\[ R_i = \frac{T_{i+1}}{T_i} \]

\[ \prod_{i=1}^{n-1} R_i = \frac{T_2}{T_1} \times \frac{T_3}{T_2} \times \ldots \times \frac{T_{n-1}}{T_{n-2}} \times \frac{T_n}{T_{n-1}} = \frac{T_n}{T_1} \]

The hardest-to-schedule task set

\[ T_1 < T_2 < \ldots < T_n < 2T_1 \]

\[ C_i = T_{i+1} - T_i \]

\[ C_n = 2T_1 - T_n = T_1 - \sum_{i=1}^{n-1} C_i \]

\[ U_i = \frac{C_i}{T_i} \Rightarrow U_i = \frac{T_{i+1} - T_i}{T_i} = \frac{T_{i+1}}{T_i} - 1 = R_i - 1 \quad 1 \leq i \leq n - 1 \]

\[ U_n = \frac{C_n}{T_n} \Rightarrow U_n = \frac{2T_1 - T_n}{T_n} = \frac{2T_1}{T_n} - 1 \]

See chapter 4.3.4 from Giorgio’s book
Deriving the hyperbolic bound

\[ U_i = \frac{C_i}{T_i} = R_i - 1 \]
\[ U_n = \frac{2T_1}{T_n} - 1 \]
\[ \prod_{i=1}^{n-1} R_i = \frac{T_n}{T_1} \]
\[ \sum_{i=1}^{n} C_i \leq T_1 \]
\[ C_n \leq 2T_1 - T_n \]

\[ \Rightarrow \frac{C_n}{T_n} \leq \frac{2T_1 - T_n}{T_n} \]
\[ \Rightarrow U_n \leq \frac{2T_1}{T_n} - 1 \]
\[ \Rightarrow U_n + 1 \leq 2 \cdot \frac{1}{\prod_{i=1}^{n-1} R_i} \]
\[ \Rightarrow (U_n + 1) \prod_{i=1}^{n-1} R_i \leq 2 \]
\[ \Rightarrow (U_n + 1) \prod_{i=1}^{n-1} (U_i + 1) \leq 2 \]
\[ \Rightarrow \prod_{i=1}^{n} (U_i + 1) \leq 2 \]
Agenda

• Secrets behind L&L and hyperbolic bound tests
• **RM schedulability analysis**
  • Response-time analysis
  • Park test
  • A test for Harmonic task sets

• Lab 2 assignment description
• **EDF schedulability analysis**
  • Optimality of EDF for periodic tasks
  • EDF response-time analysis

If you want to pass the course, learn this lecture **very well**
Previously on Real-Time Systems

Today, we introduce **exact** schedulability tests.
Exact schedulability test

If an **exact schedulability test** for **scheduling algorithm A** accepts a task set, then the task set is **certainly schedulable** by the algorithm, and if the test rejects the task set, then the task set is **certainly NOT schedulable** by the algorithm.

A task set is schedulable by algorithm A **if and only if** it is accepted by an **exact schedulability test** for algorithm A.
Response-time analysis (RTA)

Unlike utilization-based tests, response-time analysis takes task’s period and worst-case execution time (WCET) into account!

Visualization:

Worst-case response time (WCRT):

\[ R_i = \max\{R_{i,j} \mid \forall j, 1 \leq j \leq \infty\} \]

The test:

\[ \forall \tau_i \in \tau, \quad R_i \leq D_i \]

It is not practical as it is!
We need a smarter solution
For fixed-priority scheduling, the WCRT depends on the alignment of arrival times.

Task set A

Task set B
Critical instant

For any task $\tau_i$, the longest response time occurs when it arrives together with all higher-priority tasks.
Critical instant

For independent preemptive tasks under fixed priorities, the critical instant of $\tau_i$ occurs when it arrives together with all higher priority tasks.

Notations: $\tau_i(C_i, T_i, D_i, \phi_i)$
Response-time analysis (RTA) (Audsley ‘90)

\[ R_i^{(n)} = C_i + \sum_{k=1}^{i-1} \left[ \frac{R_k^{(n-1)}}{T_k} \right] \cdot C_k \]

The solution is based on **fixed-point iterations**:

Starting point:

\[ R_i^{(0)} = C_i \]

Iterate until:

\[ R_i^{(n)} \leq R_i^{(n-1)} \]

**Usage**: Use \( R_i^{(0)} \) to calculate \( R_i^{(1)} \), then use \( R_i^{(1)} \) to obtain \( R_i^{(2)} \), ..., continue until \( R_i^{(n)} = R_i^{(n-1)} \)

**Ceiling operator**: \( [x] = \min\{m \in \mathbb{Z} \mid m \geq x\} \)

Example: \( [2.4] = 3 \), \( [2.7] = 3 \), \( [-2.4] = -2 \)
Understanding the terms

\[
\begin{align*}
R_i^{(n)} &= C_i + \sum_{k=1}^{i-1} \left( \frac{R_i^{(n-1)}}{T_k} \right) \cdot C_k \\
R_i^{(0)} &= C_i \\
\text{Continue if } R_i^{(n)} > R_i^{(n-1)}
\end{align*}
\]

Make sure that the execution of \( \tau_i \) finishes

For all higher-priority tasks

Multiply that by the WCET of \( \tau_k \)

Count the maximum number of jobs released by task \( \tau_k \) in the interval \([0, R_i^{(n-1)})\)
Exercise (5min)

- Exam question: find the WCRT of tasks $\tau_1$ and $\tau_2$ when they are scheduled by rate monotonic priorities

<table>
<thead>
<tr>
<th>$\tau_i$</th>
<th>$C_i$</th>
<th>$T_i$</th>
<th>$D_i$</th>
<th>$U_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>0.4</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>4</td>
<td>10</td>
<td>10</td>
<td>0.4</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>1</td>
<td>25</td>
<td>25</td>
<td>0.04</td>
</tr>
</tbody>
</table>

\[
U = 0.84
\]

\[
\begin{align*}
R_i^{(0)} &= C_i \\
R_i^{(n)} &= C_i + \sum_{k=1}^{i-1} \left[ \frac{R_i^{(n-1)}}{T_k} \right] \cdot C_k \\
\text{Continue if } R_i^{(n)} > R_i^{(n-1)}
\end{align*}
\]
Exercise (5min)

- Exam question: find the WCRT of tasks $\tau_1$ and $\tau_2$ when they are scheduled by rate monotonic priorities

<table>
<thead>
<tr>
<th>$\tau_i$</th>
<th>$C_i$</th>
<th>$T_i$</th>
<th>$D_i$</th>
<th>$U_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>0.4</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>4</td>
<td>10</td>
<td>10</td>
<td>0.4</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>1</td>
<td>25</td>
<td>25</td>
<td>0.04</td>
</tr>
</tbody>
</table>

We stop here since $R_1^{(n)} \leq R_1^{(n-1)}$

$R_1^{(0)} = 2$

$R_1^{(1)} = 2 + \sum_{k=1}^{i-1} \left\lfloor \frac{R_1^{(0)}}{T_k} \right\rfloor \cdot C_k = 2$

WCRT of $\tau_1$ is 2
Exercise (5min)

- Exam question: find the WCRT of tasks $\tau_1$ and $\tau_2$ when they are scheduled by rate monotonic priorities

<table>
<thead>
<tr>
<th>$\tau_i$</th>
<th>$C_i$</th>
<th>$T_i$</th>
<th>$D_i$</th>
<th>$U_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>0.4</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>4</td>
<td>10</td>
<td>10</td>
<td>0.4</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>1</td>
<td>25</td>
<td>25</td>
<td>0.04</td>
</tr>
</tbody>
</table>

\[
R_i^{(n)} = C_i + \sum_{k=1}^{i-1} \left\lfloor \frac{R_i^{(n-1)}}{T_k} \right\rfloor \cdot C_k
\]

\[
R_i^{(0)} = C_i
\]

Continue if $R_i^{(n)} > R_i^{(n-1)}$

$R_2^{(0)} = 4$

\[
R_2^{(1)} = 4 + \sum_{k=1}^{2-1} \left\lfloor \frac{R_2^{(0)}}{T_k} \right\rfloor \cdot C_k = 4 + \left\lfloor \frac{4}{5} \right\rfloor \cdot 2 = 6
\]

$R_2^{(1)} \leq R_2^{(0)}? \quad \text{NO, so continue}$

\[
R_2^{(2)} = 4 + \sum_{k=1}^{2-1} \left\lfloor \frac{R_2^{(1)}}{T_k} \right\rfloor \cdot C_k = 4 + \left\lfloor \frac{6}{5} \right\rfloor \cdot 2 = 8
\]

$R_2^{(2)} \leq R_2^{(1)}? \quad \text{NO, so continue}$

\[
R_2^{(3)} = 4 + \sum_{k=1}^{2-1} \left\lfloor \frac{R_2^{(2)}}{T_k} \right\rfloor \cdot C_k = 4 + \left\lfloor \frac{8}{5} \right\rfloor \cdot 2 = 8
\]

$R_2^{(3)} \leq R_2^{(2)}? \quad \text{Yes, so stop}$

$\quad \rightarrow \text{WCRT of } \tau_2 \text{ is 8}$
Exercise (5min)

- Exam question: find the WCRT of tasks \( \tau_1 \) and \( \tau_2 \) when they are scheduled by rate monotonic priorities

<table>
<thead>
<tr>
<th>( \tau_i )</th>
<th>( C_i )</th>
<th>( T_i )</th>
<th>( D_i )</th>
<th>( U_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_1 )</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>0.4</td>
</tr>
<tr>
<td>( \tau_2 )</td>
<td>4</td>
<td>10</td>
<td>10</td>
<td>0.4</td>
</tr>
<tr>
<td>( \tau_3 )</td>
<td>1</td>
<td>25</td>
<td>25</td>
<td>0.04</td>
</tr>
</tbody>
</table>

\[
R_i^{(n)} = C_i + \sum_{k=1}^{i-1} \left[ \frac{R_i^{(n-1)}}{T_k} \right] \cdot C_k \\
R_i^{(0)} = C_i \\
\text{Continue if } R_i^{(n)} > R_i^{(n-1)}
\]

\( R_3^{(0)} = 1 \)

\[
R_3^{(1)} = 1 + \sum_{k=1}^{2} \left[ \frac{R_3^{(0)}}{T_k} \right] \cdot C_k = 1 + \left[ \frac{1}{5} \right] \cdot 2 + \left[ \frac{1}{10} \right] \cdot 4 = 7
\]

\( R_3^{(1)} \leq R_3^{(0)}? \) \ NO, so continue

\[
R_3^{(2)} = 1 + \sum_{k=1}^{1} \left[ \frac{R_3^{(1)}}{T_k} \right] \cdot C_k = 1 + \left[ \frac{7}{5} \right] \cdot 2 + \left[ \frac{7}{10} \right] \cdot 4 = 9
\]

\( R_3^{(2)} \leq R_3^{(1)}? \) \ NO, so continue

\[
R_3^{(3)} = 1 + \sum_{k=1}^{0} \left[ \frac{R_3^{(2)}}{T_k} \right] \cdot C_k = 1 + \left[ \frac{9}{5} \right] \cdot 2 + \left[ \frac{9}{10} \right] \cdot 4 = 9
\]

\( R_3^{(3)} \leq R_3^{(2)}? \) \ Yes, so stop

\[\rightarrow \text{WCRT of } \tau_3 \text{ is 9}\]
### Visualizing the WCRT

<table>
<thead>
<tr>
<th>(\tau_i)</th>
<th>(C_i)</th>
<th>(T_i)</th>
<th>(D_i)</th>
<th>(U_i)</th>
<th>(R_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\tau_1)</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>0.4</td>
<td>2</td>
</tr>
<tr>
<td>(\tau_2)</td>
<td>4</td>
<td>10</td>
<td>10</td>
<td>0.4</td>
<td>8</td>
</tr>
<tr>
<td>(\tau_3)</td>
<td>1</td>
<td>25</td>
<td>25</td>
<td>0.04</td>
<td>9</td>
</tr>
</tbody>
</table>

\[
R_i^{(n)} = C_i + \sum_{k=1}^{i-1} \left[ \frac{R_i^{(n-1)}}{T_k} \right] \cdot C_k \\
R_i^{(0)} = C_i \\
\text{Continue if } R_i^{(n)} > R_i^{(n-1)}
\]

An iteration is needed whenever \(R_i^n\) becomes larger than the arrival time of a new higher-priority job that has not been considered in \(R_i^{n-1}\)

What is \(R_{3,2}\)?

Why is \(R_{3,2}\) smaller than \(R_3\)?
**Busy-window level** $i$

$[0, 9)$ is a busy-window level 3

$[0, 8)$ is a busy-window level 2

$[0, 2)$ is a busy-window level 1

A **busy-window level** $i$ is a window of time during which the processor is busy by executing tasks with priority $P_i$ or higher.

\[
\begin{align*}
R_i^{(n)} &= C_i + \sum_{k=1}^{i-1} \left[ \frac{R_i^{(n-1)}}{T_k} \right] \cdot C_k \\
R_i^{(0)} &= C_i \\
\text{Continue if } R_i^{(n)} > R_i^{(n-1)}
\end{align*}
\]
Computational complexity of RTA

How many iterations are needed to calculate $R_3$?

$R_3^{(0)} = 3$  $R_3^{(1)} = 9$  $R_3^{(2)} = 11$

$R_3^{(3)} = 17$  $R_3^{(4)} = 19$  $R_3^{(5)} = 19$

The stop condition basically ensures that the workload that must be finished will be finished before a new task arrives.

<table>
<thead>
<tr>
<th>$\tau_i$</th>
<th>$C_i$</th>
<th>$T_i$</th>
<th>$D_i$</th>
<th>$U_i$</th>
<th>$R_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>0.4</td>
<td>2</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>4</td>
<td>10</td>
<td>10</td>
<td>0.4</td>
<td>8</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>3</td>
<td>25</td>
<td>25</td>
<td>0.12</td>
<td>19</td>
</tr>
</tbody>
</table>
Computational complexity of RTA

- Calculating the WCRT of a task using RTA is a **weakly NP-Hard** problem [Eisenbrand2008]

- It has a pseudo-polynomial time computational complexity

  What does it mean?

  Example: $O(w \cdot n)$
  - $n$ is the number of tasks
  - $w$ is the ratio of the largest to the smallest period

  $w = 2^n$
  \[ \Rightarrow O(w \cdot n) = O(n \cdot 2^n) \rightarrow \text{exponential} \]

[wiki]: In computational complexity, an NP-complete (or NP-hard) problem is **weakly NP-complete** (or weakly NP-hard), if there is an algorithm for the problem whose running time is polynomial in the dimension of the problem and the magnitudes of the data involved (provided these are given as integers), rather than the base-two logarithms of their magnitudes.

Computational complexity of RTA

• Calculating the WCRT of a task using RTA is a weakly NP-Hard problem [Eisenbrand2008]

• It has a pseudo-polynomial time computational complexity
  
  • The complexity not only depends on the number of tasks $n$ but also depends on the periods and execution times

  • Namely, for some task sets it is very fast, and for some other task sets with the same number of tasks, it is very very slow

Don’t worry.
In practice, RTA is really fast for periodic tasks :D

[wiki]: In computational complexity, an NP-complete (or NP-hard) problem is weakly NP-complete (or weakly NP-hard), if there is an algorithm for the problem whose running time is polynomial in the dimension of the problem and the magnitudes of the data involved (provided these are given as integers), rather than the base-two logarithms of their magnitudes.

How to make the RTA faster?

\[
R_i^{(n)} = C_i + \sum_{k=1}^{i-1} \left[ \frac{R_i^{(n-1)}}{T_k} \right] \cdot C_k \\
R_i^{(0)} = C_i \\
\text{Continue if } R_i^{(n)} > R_i^{(n-1)}
\]

Instead of starting from

\[
R_i^{(0)} = C_i
\]

Start from

\[
R_i^{(0)} = R_{i-1} + C_i
\]

Why is this okay?

Because anyway \( \tau_i \) cannot start its execution until the busy-window level \( i - 1 \) has finished!

A better starting point!

What do you suggest?
Using RTA to build fast sufficient tests

Park’s test

\[
\begin{align*}
R_i^{(n)} &= C_i + \sum_{k=1}^{i-1} \left( \frac{R_i^{(n-1)}}{T_k} \right) \cdot C_k \\
R_i^{(0)} &= C_i \\
\text{Continue if } R_i^{(n)} &> R_i^{(n-1)}
\end{align*}
\]

Idea: Instead of searching for the exact WCRT, just check if the workload that must be completed before the deadline is smaller than the deadline of the task.

How?

Using RTA to build fast sufficient tests

Park’s test

\[
\begin{align*}
R_i^{(n)} &= C_i + \sum_{k=1}^{i-1} \left[ \frac{R_i^{(n-1)}}{T_k} \right] \cdot C_k \\
R_i^{(0)} &= C_i
\end{align*}
\]

Continue if \( R_i^{(n)} > R_i^{(n-1)} \)

Idea:
Instead of searching for the exact WCRT, just check if the workload that must be completed before the deadline is smaller than the deadline of the task

workload that must be completed before the deadline

\[
C_i + \sum_{k=1}^{i-1} \left[ \frac{D_i}{T_k} \right] \cdot C_k \leq D_i
\]

What is the computational complexity per task?

\( O(n) \)

Why the Park’s test is just a sufficient test?

Park’s test is just a sufficient test

Is this task set schedulable by RM?
Yes! The critical instant is schedulable.
=> The WCRT of each task is smaller than its deadline

What is the WCRT of $\tau_3$ and $\tau_4$?
8 and 9, respectively.

What does Park’s test say about $\tau_4$?
It says $\tau_4$ will miss its deadline

Why did it happen?
When a higher-priority task is released after the WCRT of the task under study, Park’s test considers that task within the workload that must be finished by the deadline! Namely, it upper-approximate the workload!

$$C_4 + \sum_{k=1}^{4-1} \left[ \frac{D_k}{T_k} \right] \cdot C_k \leq D_4 \Rightarrow 1 + \left( \left\lfloor \frac{10}{5} \right\rfloor \cdot 2 \right) + \left( \left\lfloor \frac{10}{9} \right\rfloor \cdot 3 \right) + \left( \left\lfloor \frac{10}{10} \right\rfloor \cdot 1 \right)$$

$$= 1 + 4 + 6 + 1 = 12 \not\leq 10$$
A test for harmonic tasks

Han et al. [1997] have proven that rate monotonic is optimal if periods are harmonic (each period divides smaller ones).

Hence, if periods are harmonic, the following test is both necessary and sufficient for RM schedulability

$$U \leq 1$$

Agenda

• Secrets behind L&L and hyperbolic bound tests
• RM schedulability analysis
  • Response-time analysis
  • Park test
  • A test for Harmonic task sets

• Lab 2 assignment description
• EDF schedulability analysis
  • Optimality of EDF for periodic tasks
  • EDF response-time analysis

If you want to pass the course, learn this lecture very well
Lab 2 is about implementing an online scheduling algorithm

- **Your mission**
  - Re-write the code for the scheduler and timer interrupt handler so that the scheduler is called at most only once per job-release event and at most only once per job-completion event.
Lab 2 is about implementing an online scheduling algorithm

- **Expected deliverables**
  - Your code! It must be without bug and you should demo it to the TA.
  - A description of your solution
  - A plot that shows the “percentage of overheads in one hyperperiod” as a function of the number of tasks in the system for your implementation
Challenge questions

**Challenge 1: Efficient handling of timer events**

- **The problem?**
  - There are multiple periodic tasks but only one hardware timer!
  - Going through all tasks to see which of them has a job to release will take a lot of time if you want to do it in every timer event.

- **Solution**
  - Use better data structures to handle “next timer events” such as a timer wheel!

---

**Figure 5. Overhead battle.**

Challenge questions for Lab 2 (not mandatory)

This part describes some challenges for those who enjoyed this assignment so far and are willing to take a bigger challenge to expand their experiences with practical aspects of real-time system design. If your response to this part is correct, you will directly receive your bonus points gained in this part on top of your final grade.

Challenge 1 (0.4 points for the final grade)

As you may have noticed, in the design proposed in Figure 1, each time the timer interrupt is called (namely, a job is released), you need to traverse through the list of tasks and update their next arrival time in case they are released. This is needed because at some points in time, there might be more than one task that release a job, e.g., at times 0 and 14 in our example. Moreover, you also need to find out when the next release event happens, hence, you need to find the minimum value among the next release times and set the timer to that value. In an inefficient implementation, you may traverse through the whole list of tasks to perform these two functions.

Goal. The goal of this challenge question is to reduce the number of times that you go through the whole list whenever the timer interrupt is called. More formally, we want to reduce the average overhead of handling job-release events.

Why should you care about your timer interrupt handler? Many real-time systems use harmonic periods\(^2\). For example, in Automotive industry\(^3\), AutoSAR standard forces the manufacturers to use the following set of periods: \(1, 2, 5, 10, 20, 50, 100, 200, 1000\)ms. As you can see, the task with the smallest period will release 1000 more jobs every second than the task with the largest period. Moreover, automotive systems usually have a large set of tasks (e.g., a couple of hundred tasks). As a result, if your design requires traversing over the list of tasks each time the task with period 1ms releases a job, the system will be crashed down by the overhead (it is a deadly situation for an engineer)! Taking into account that you are working with a slow microprocessor, overheads of a bad implementation can easily kill your whole system\(^4\).
Challenge questions

Challenge 2: Handle overflow of time variables!

• The problem?
  • Your value overflows and your solution is doomed!
  • If you keep increasing time values, soon you will face an overflow

• Solution
  • Try to reset time values at the end of hyperperiod

Challenge 2 (0.2 points for the final grade)

As you may have experienced in the past, if you continue adding up values to a variable, it will eventually overflow. This can easily happen for your timer interrupt handler if you keep the next release time of a job as an absolute value.

Why should you care about overflows? Because they make your boss and your customers extremely angry! To observe the effect of overflow in your program, try to plot the schedule of 10 hyperperiods and see how your schedule becomes unexplainable! Never under-estimate overflows.

Goal. The goal of this challenge is to implement a solution in which there is no overflow for timer events. More formally, you will implement a solution that can schedule task sets for the rest of the life time of the system.

A possible solution approach. One possible solution would be to keep track of the time instants (for example, job-release times) in the form of “relative” values with respect to the beginning of the hyperperiod. This provides you the opportunity to reset the time values back to zero at the end of each hyperperiod and be prepared for the next one. By doing this, the first job of \( t_j \) in Figure 1 will always be released at time 0 and the second job of \( t_j \) will always be released at time 7 because you conceptually reset the time at the beginning of the hyperperiod.

Can you guess what is a pre-requisite for the soundness of this solution? Yes! The hyperperiod itself must be smaller than the time at which an integer variable in your system overflows (e.g., smaller than 65,535 clock ticks), otherwise you will get an overflow within your first hyperperiod.

If you have been tempted to use “\textit{mod}” instruction\(^1\) to ensure that every variable stays within the boundary of the hyperperiod, you should be aware of the overhead of calculating \( \% \) in the MSP430 microcontroller. Hence, try to come up with a solution that does not need \% operation.

The above-mentioned solution is one of the possible solutions. Feel free to come up with your own! :-)

Deliverables.

• Hand-in your code for this challenge in a separate zip file but at the same time that you hand-in your main assignment for Lab 2. Be prepared to demo your code to the TA.
• Briefly explain your design and discuss how you have avoided overflows. Your answer can include figures/tables. It should not have more than 500 words.
• Draw a plot that shows your solution can schedule tasks for long duration of time (e.g., for 20 sec).